Chapter 4

Texture Mapping

Reading:
Angel’s Interactive computer graphics, 6th ed.
Sections 7.4~7.10
Outline

- Texture mapping
- 2D-to-3D texture mapping
- Anti-aliasing
- 3D Solid texture mapping
- Environment mapping
- Bump map
Texture mapping

• Limitations of pipeline graphics
  – **Modeling complexity**
    • Modeling lumpy surfaces or surface of complicated color
  – **Rendering complexity**
    • Limitation of vertex lighting and local lighting in pipeline graphics

• Usefulness of texture mapping
  – Adds surface and attribute details
  – Adds lighting effects
Texture mapping

• Considerations
  – What attribute or parameter of the model is to be modulated to produce the desired texture effect?
    • Diffuse color, specular coefficients
    • Surface normal perturbation, position displacement
    • Lighting or reflection result

• Texture mapping tends to produce worse aliasing artifacts than image synthesis techniques.
  – Requires special anti-aliasing treatments
Types of texture mapping

• 2D to 3D mapping
  – Color map: Modulates the surface color
  – Other texture maps
    • Bump mapping, displacement mapping
      – Modulates surface normal or position
    • Shadow mapping, normal map, gloss map, ....
    • Environment mapping, light map

• 3D texture mapping
  – Avoids 2D texture mapping generation
Example - 1

Gouraud shading

Phong shading

Phong shading with Bump map
Example - 2

geometric model

texture mapped
Example - 3

Environment map
Example - 4

Xbox game: Project Gotham Racing
Example - 5

- Combination of normal map and environment map is particularly effective
Example - 6

Bump map

Displacement map
Example - 7

Bump map

Color map

(From 3D Games by Watt et al.)
Example - 8

Gloss Map Example

Shiny puddles

Specular lighting contribution (per-vertex lighting) × (modulate) Gloss map texture + Diffuse lighting contribution (per-vertex lighting) = Final combined result
Example - 9

Without Light map (no shadow, Global lighting)

[From Channa’s article]
Example - 10

The white rhombus type object (on the right hand side) represents a point light source.
2D to 3D texture mapping

- **Global texture mapping (per vertex)**
  - 2D texture pattern $T(s, t)$, called texels
  - Associate a unique value of $T(s, t)$ with each vertex on a 3D mesh
    - Done in modeling stage
      - Require mesh parameterization

- **Local Texture mapping (per-pixel)**
  - Associate a texture value of $T(s, t)$ with each pixel within the polygon on the screen
    - Require so called perspective-correction interpolation
  - **A side note**
    - In fact, a pixel corresponds to an area of the texture map (area-to-area mapping)
2D to 3D texture mapping
Problems -1

- 2D to 3D mapping is in general hard
  - Mapping 3D surface to a 2D texture domain is generally complex and have undesirable properties such as distortion
  - Nature of rendering process
    - 2D-to-3D mapping in pipeline is done by a pixel-by-pixel process
      - Each pixel is associated with a texture coordinate
      - But the pixel-to-texture mapping is area-to-area mapping
        » Impossible to find the texels corresponding to each visible pixel in the pipeline process
    - Must deal with aliasing carefully
2D to 3D texture mapping
Problems -2

Pixel -> an area on surface -> an area on the texture.
This is an area-to-area mapping.
2D to 3D texture mapping

Texture Mapping for a polygon mesh

• Global mapping
  – How do we map each vertex into texture space?
    • Most popular: associate \((u, v)\) to polygon vertices in modeling phase (by users)
      – by UV mapping
      – Use forward or inverse two-stage mapping (simple)
      – In general by mesh parameterization

• Local mapping
  – In rasterization, how do we compute pixel’s texture coordinate?
    • Bilinear interpolation is incorrect for perspective proj.
  – How to access texture map with the texture coordinate?
2D to 3D texture mapping
Global Mapping

"Zookeeper"
by Tito Pagan
10.19.05

[Tito Pagan]
Two-Stage Mapping

- Can take care of both local and global mapping simultaneously.
- Two stages: (a forward mapping)
  - S-mapping: maps from texture space to simple intermediate surface such as a cylinder or sphere
    \[ T(u,v) \rightarrow T'(x', y', z') \]
  - O-mapping: maps points on the intermediate surface onto the object surface
    \[ T'(x',y',z') \rightarrow O(x,y,z) \]
- Can distort the texture pattern onto the object.
Two-Stage Mapping

(a) 2D texture map

(b) Intermediate surface

Object
S-Mapping

- Intermediate surface is context-dependent and in general possesses an analytic representation, e.g.,
  - A plane at any orientation.
  - The (interior) surface of a cylinder.
  - The (interior) face of a cube.
  - The (interior) surface of a sphere.
O-Mapping

1. The intersection of the reflected view ray with the intermediate surface (environmental mapping)

2. The intersection of the surface normal at \((x, y, z)\) with \(T\). (inverse mapping)
3. The intersection of a line through 
\((x, y, z)\) and the object centroid with \(T'\).
(inverse mapping)

4. The intersection of a line from 
\((x, y, z)\) to \(T'\) whose orientation 
is the normal at \((x', y', z')\).
\(((x', y', z') \rightarrow (x, y, z))\)
Cylindrical Mapping

\[(r \cos \theta, r \sin \theta, h)\]

\[0 < \theta < 2\pi, \ 0 < h < 1\]

\[(u, v) = (\theta / 2\pi, h)\]
Spherical Mapping

- Have to take care of the distortion at the poles

\[ (r \cos \theta \sin \phi, r \sin \theta \sin \phi, r \cos \phi) \]
\[ 0 < \theta < \pi/2, \pi/4 < \phi < \pi/2, \]
\[ (u, v) = (2 \theta / \pi, ((\pi/2) - \phi) / (\pi/4)) \]
Box mapping

![Diagram showing box mapping with axes labeled s and t, and regions labeled Front, Back, Bottom, Right, and Top.](image-url)
Forward and inverse mapping

Forward mapping
- May produce holes and overlaps in texture images.

Inverse mapping
- Easily to be incorporated into a Z-buffer algorithm.
- Is preferred in practice.
Forward and inverse mapping - 2

Forward two-stage mapping

2D texture map

(a) Intermediate surface

(b) Object

Inverse two-stage mapping
Inverse two-stage mapping
Shrinkwrap mapping - 1

- Inverse map 4 pixel corners to 4 points $(x_w, y_w, z_w)$ on the object surface.
- Apply O-mapping to find the point $(\theta, h)$ on the cylinder surface. In the shrinkwrap case, join the center of the cylinder to the object point $(x_w, y_w, z_w)$ and intersect this line with the cylinder surface. This gives us $(x', y', z')$. $(\theta, h)$ is obtained by
  
  $$(x_w, y_w, z_w) \rightarrow (\theta, h) = (\tan^{-1}(y_w / z_w), z_w)$$

- Apply S-mapping to find $(u, v)$ corresponding to $(\theta, h)$. 

Inverse two-stage mapping
Shrinkwrap mapping - 2
Examples

A plane

A cylinder

A sphere
2D to 3D texture mapping
Local mapping

- **Perspective-correct interpolation**
  - **Affine texture mapping**
    - Texture coordinates are interpolated at the same time as interpolating color and depth in image space
    - Because bilinear interpolation does not take into account the depth information about a polygon's vertices, for the perspective projection it produces a noticeable defect.
Perspective-correct interpolation

Wire-frame view

Textured view
Perspective-correct interpolation

Affine mapping: Preserve line and parallelsm

Projective mapping: Preserve line only.
Parallel lines tend to Merge in the distance
Perspective-correct interpolation

Figure 1: Straightforward linear interpolation of attribute values in the screen space (or in the image plane) does not always produce perspective-correct results.
Perspective-correct interpolation

– **Instead of interpolating the texture coordinates directly,**
  - the coordinates are divided by their **depth** (relative to the viewer), and
  - the reciprocal of the depth value is also interpolated and used to recover the perspective-correct coordinate.
Perspective-correct interpolation

- **Problem:** we don’t know $z$ anymore
- **Solution:** we do know $w = -z$
  - Vertex coordinate $r = (p_x, p_y, p_z, 1/w, u/w, v/w)$
    - $(p_x, p_y, p_z)$ is the point after perspective division
    - linearly interpolated $1/w$ and $(u/w, v/w)$
    - The interpolated texture coordinates are divided by the interpolated $1/w$ to get the perspective-correct coordinate.

$$r_0 = \begin{bmatrix} p_0, \frac{1}{w_0}, u_0, v_0 \end{bmatrix}$$

$$r_1 = \begin{bmatrix} p_1, \frac{1}{w_1}, u_1, v_1 \end{bmatrix}$$

$$r_2 = \begin{bmatrix} p_2, \frac{1}{w_2}, \frac{u_2}{w_2}, \frac{v_2}{w_2} \end{bmatrix}$$

$$r_3$$

$$r_4$$

Interpolated values

$$(u, v) = \left( \frac{u}{w}, \frac{v}{w} \right) / \left( \frac{1}{w} \right)$$
Perspective-Correct Interpolation

Summary

\[
t = \frac{sZ_1}{sZ_1 + (1-s)Z_2}
\]

Interpolating z-values

\[
Z_t = Z_1 + t(Z_2 - Z_1)
\]

Interpolating attribute values

\[
I_t = I_1 + t(I_2 - I_1)
\]

Figure 2: The virtual camera is looking in the +z direction in the camera coordinate system. The image plane is at a distance of \(d\) in front of the camera. A, B and C are points on the primitive with attribute values \(I_1\), \(I_2\) and \(I_t\) respectively, and their images on the image plane are \(a\), \(b\) and \(c\), respectively. \(s\) and \(t\) are parameters used for linear interpolation.
\[
\frac{X_1}{Z_1} = \frac{u_1}{d} \Rightarrow X_1 = \frac{u_1 Z_1}{d},
\]

\[
\frac{X_2}{Z_2} = \frac{u_2}{d} \Rightarrow X_2 = \frac{u_2 Z_2}{d},
\]

\[
\frac{X_t}{Z_t} = \frac{u_s}{d} \Rightarrow Z_t = \frac{d X_t}{u_s}. 
\]

\[
\begin{aligned}
& u_s = u_1 + s(u_2 - u_1) \\
& X_t = X_1 + t(X_2 - X_1), \\
& Z_t = Z_1 + t(Z_2 - Z_1), 
\end{aligned}
\]

\[
Z_t = \frac{d}{u_1 + s(u_2 - u_1)} \left( \frac{u_1 Z_1 + t(u_2 Z_2 - u_1 Z_1)}{d} \right)
\]

\[
= \frac{u_1 Z_1 + t(u_2 Z_2 - u_1 Z_1)}{u_1 + s(u_2 - u_1)}.
\]

\[
Z_1 + t(Z_2 - Z_1) = \frac{u_1 Z_1 + t(u_2 Z_2 - u_1 Z_1)}{u_1 + s(u_2 - u_1)}.
\]

\[
t = \frac{s Z_1}{s Z_1 + (1 - s) Z_2} (Z_2 - Z_1)
\]
Antialiasing
OpenGL basic texture access

- GL_NEAREST
  - Get weighted average color

- GL_LINEAR
  - Get weighted average color
Antialiasing

• Visible artifacts come from the incorrect sum on weighted value of texels that fall within the pre-image
  – Area-to-area mapping vs. point sampling

• Aliasing is extremely problematic in texture mapping, especially in texture with coherence or periodicity

• Antialiasing methods
  – Properly sum weighted values of texels inside the pre-image
  – Mip-mapping
    • The most common method
    • Pre-calculates the sum based on an assumption that the pre-image is reasonably close to a square.
Aliasing problem

World Space (Top View)  Screen Space (Front View)  Texture Space
Aliasing problem

As object moves away from the viewer, its projection in screen become smaller, and the pixel’s pre-image becomes larger.

Move away from viewer

Object size decreases

Pixel pre-image increase

Pre-image

Inverse mapping

With anti-aliasing

Without anti-aliasing

Pixel shade

Anti-aliasings. Point sampling
Mip-Mapping

- **Minification**
  - When an object moves away and becomes small in screen a pixel has a larger pre-image in texture space (since texture is fixed), so many texels map into a single pixel.
  - A pixel maps onto many texels.

- **Magnification**
  - When an object becomes close to the viewer and only part of it may occupy the whole screen, resulting in pixel pre-images that have less area than one texel.
  - A pixel maps onto less one texel.
Mip-Mapping - 2

Pre-image: curvilinear quadrilateral

minification (c)

Pre-image: approx. by a square

magnification (d)
Mip-Mapping

• Mip-map
  – A set of pre-filtered texture maps, each is exactly half the resolution of the previous one. Thus each texel in a low resolution map represents the average of a number of texels from the previous map.

• Selection
  – An object near to the viewer, and larger in screen, selects a single texel from a high resolution map.
  – An object further away from the viewer and smaller in screen, selects a single texel from a low resolution map.
Mip-Mapping

Level 0

Level 1

Level 2

\( d \text{ axis} \)
Mip-Mapping - 5
Mip-Mapping

- By a suitable choice of $D$, an image at appropriate resolution is selected
  - To avoid discontinuities between the images at varying resolutions, texture access from different levels are blended.
  - Tri-linear interpolation
    - The images are discontinuous in resolution.
    - $D$ is a continuous parameter.
    - Linear interpolation is carried out for texture result from two nearest levels.
      - For each level, linear interpolation can be applied for minification cases
Mip-Mapping

How to incorporate mip-mapping with Gouraud shading?

- Vertex-texture-coordinate assignment is given (for original texture map).
- At rendering (inside the pipeline)
  - Determine an appropriate level, or two nearest levels according to D
  - Polygon shading (for a level)
    - Adjust vertex’s texture coordinates (due to resolution change)
    - Compute pixel’s texture coordinate by interpolation
    - Obtain texture value using specified filtering methods, such as GL_nearest, GL_linear, or their combination.
    - Interpolate texture results if two levels are used
    - Modulate the image pixel
Mip-Mapping

- Mip-mapping results in overblurring
  - occurs when a pixel’s pre-image covers a large number of texels in $u$ direction and only a few in $v$ direction. To avoid aliasing, mip-map of the lowest resolution (indicated by longer edge of quad) is selected, and but resulting in overblurring.

- Approaches [Real Time Rendering (2nd ed.), p 137]
  - Rectangular filtering (anisotropic filtering)
    - Ripmap [Advanced OpenGL course notes 98]
      - Extend mip-map to include downsampled rectangular areas as subtextures that can be accessed.
      - Used in high-end HP graphics accelerator in early ’90.
    - Crow’s summed-area table [Crow, Siggraph84]
      - Never implemented in hardware
    - Both are memory intensive.
  - Unconstrained anisotropic filtering (available in OpenGL)
Example
Point sampling with nearest neighbor

Mip-mapping (overblurring)

Summed area table
Example

- **Left:** without AF
- **Right:** with AF
Texture mapping in OpenGL

• OpenGL supports
  – 1D and 2D textures to 1~4D graphical objects.
  – 3D TM: available in high-end hardware.
  – We focus on 2D-to-2D mapping.

• OpenGL’s TMs rely on the pipeline arch.
  – Geometric pipeline and pixel pipeline
  – TM is applied when primitives are rasterized.
  – TM is part of the shading process.
    • Vertices are mapped to texture coordinates.
    • Texture values are obtained via interpolation.
Texture mapping in OpenGL -2

Vertices → Geometric processing → Rasterization → Display

Pixels → Pixel operations
Texture mapping in OpenGL

- **A 2D texture is specified through**
  ```
  glTexImage2D(GL_TEXTURE_2D, level, components, width, height, border, format, type, t-array);
  ```

- **Enable texture mapping**
  ```
  glEnable(GL_TEXTURE_2D)
  ```

- **Specify how the texture is mapped**
  ```
  glBegin(GL_QUAD);
  glTexCoord2f(0.0, 0.0);
  glVertex2f(x1, y1, z1);
  glTexCoord2f(1.0, 0.0);
  glVertex2f(x2, y2, z2);
  glTexCoord2f(1.0, 1.0);
  glVertex2f(x3, y3, z3);
  glTexCoord2f(0.0, 1.0);
  glVertex2f(x4, y4, z4);
  glEnd();
  ```
Problems of TM in OpenGL

• Many formats for color images and many different ways to store the bit patterns for each color.

• What happen if we specify a value of s or t outside of the range (0.0, 1.0).
  – To repeat or to clamp the value to 0.0 or 1.0 by
    ```
    glTexParameteri(GL_TEXTURE_WRAP_S, GL_REPEAT/GL_CLAMP)
    glTexParameteri(GL_TEXTURE_WRAP_T, GL_REPEAT/GL_CLAMP)
    ```
Problems of TM in OpenGL

• Aliasing
  – Due to magnification and minification

Antialiasing:
  – Use the value of nearest texel
    
    \[
    \text{glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_NEAREST)}
    \]
    
    \[
    \text{glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_NEAREST)}
    \]
  
  – Use filtering to obtain a weighted average texel value in the neighborhood of the nearest texel. OpenGL uses 2x2 average:
    
    \[
    \text{glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR)}
    \]
Problems of TM in OpenGL

- **Mipmapping antialiasing for minification**
  - A series of texture arrays at reduced size.
    
    ```
    gluBuild2DMipmaps(GL_TEXTURE_2D, 3, 64, 64, GL_RGB, GL_UNSIGNED_BYTE, tex-array);
    ```
  - Mipmaps are invoked automatically after specifying
    
    ```
    gluTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST_MIPMAP_NEAREST);
    ```
Problems of TM in OpenGL

- Interaction between texture and shading
  - Modulate the shade (by multiplication)
    \[
    \text{glTexEnv(GL\_TEX\_ENV, GL\_TEX\_ENV\_MODE, GL\_MODULATE);}\]
  - Replace the shade
    \[
    \text{glTexEnv(GL\_TEX\_ENV, GL\_TEX\_ENV\_MODE, GL\_DECAL);}\]
Problems of TM in OpenGL

- Proper TM depends on the type of projection
  - OpenGL uses linear interpolation in screen space to find a texture value.
  - Orthogonal projection: Correct
  - Perspective projection: Not correct due to the nonlinear depth scaling. OpenGL can employ a better interpolation at a time penalty by

\[
glHint(GL_PERSPECTIVE_CORRECTION, GL_NICEST);\]
Advanced features in OpenGL

• Add a border around the texture
  – to match edges between multiple maps.

• Automatically generate texture coord.
  – Can form a linear map between geometric coord. to texture coord.
  – Useful in terrain rendering.

• Texture objects
  – Allows application program to define texture maps and retain them in memory even though the present texture changes.
Environment Mapping

• Reflecting a surrounding environment in a shiny object.
  – Can be effectively handled by expensive ray tracing.

• As a cheap alternative, we pre-store the surrounding environment as a map.
  – Approximate ray-tracing with only one reflection ray.
  – Two-stage mapping: a view-independent map generation, followed by a view-dependent mapping.
Environment Mapping

Environment map on the surface of a sphere

View point

Object

(a) Area subtended in environment map
Environment Mapping

- Map creation (view-independent)
  - Uses a box as an intermediate surface.
  - Takes six photographs, or rendering six maps
  - Spherical map suffers from distortion at poles.

- Mapping operation (view-dependent)
  - Fire four rays (from eye) through the pixel’s corners, defining a reflection cone,
  - Filter the region that subtends the in the environment map to give a single shaded attribute for the pixel.

- Environment map v.s. inverse mapping
- Environment map and panoramic viewing
Environment Mapping

(a) Area subtended in environment map

(b) Object surface

View point

Pixel

(c) Cubic environment map in 2D space
Environment Mapping in OpenGL

• Map creation
  – Construct the environment map as usual.
  – Find the texture coordinate for each vertex using inverse O-mapping (view-independent).

• Rendering and mapping operation
  – OpenGL texture mapping.
  – Produce image with reflection that is view-independent.
An Example
A Comparison to Ray Tracing
Example - 1

Xbox game: Project Gotham Racing
Example - 2

- Combination of normal map and environment map is particularly effective
Bump Mapping

- Through variations in surface normal
  - Surface normal is perturbed according to a 2D bump map.
  - This tricks a local reflection model into producing local geometric variations on a smooth surface (without the need to geometrically model these depressions).

- Bump map
  - a height field and its height variation is transferred into orientation perturbations.
Bump Mapping

- $P(u)$
  - Original Surface

- $B(u)$
  - A bump map

- $P'(u)$
  - Lengthening or shortening $O(u)$ using $B(u)$

- $N'(u)$
  - The vectors to the new surface
Bump Mapping

- Let $P(u, v)$ be a parametric surface
  - Perturb the normal $n$ to $n'$
  - $n'$ is the normal of displaced surface $P'$
  - The displacement must lie on the tangent plane of $P$
  - Two arrays that contain partials of $B(u, v)$ can be precomputed.

$$N(u, v) = \frac{P_u \times P_v}{|P_u \times P_v|}$$

$$P'(u, v) = P(u, v) + B(u, v)N(u, v)$$

$$N'(u, v) = P'_u (u, v) \times P'_v (u, v)$$

where

$$P'_u = P_u + B_u N + BN_u$$

$$P'_v = P_v + B_v N + BN_v$$

$$N' = N + \left[ B_u N \times P_v + B_v P_u \times N \right]$$

$$= N + \left[ B_u N \times P_v - B_v N \times P_u \right]$$

$$= N + \left[ B_u A - B_v B \right] = N + D$$

$A$ and $B$ are on the tangent plane of $P(u, v)$. 
Bump Mapping

Surface normal at point $P$
$N = P_y \times P_u$

$D$ is given by
$D = B_u A - B_v B$
Bump Mapping

Original surface normals

Perturbed surface normals

Bump map $B(u, v)$

Shaded surface

$N' = N + D$
Bump Mapping

- For polygonal mesh
  - A multi-pass technique that can exploit standard texture mapping hardware.
  - See Programming with OpenGL: Advanced Rendering. From
    
    http://www.sgi.com/software/opengl/advanced97/notes/
Example -1
Example -2

Phong shading with Bump map

Gouraud shading
Phong Shading with Bump Map texture
Multitexturing

- Two or more textures are applied in a single rendering pass
  - Hardware supported to render in a single pass
  - Multiple textures can be fetched when processing a single fragment
  - Draw object once with several textures, some of which can be dynamically modified. E.g.,
    - The texture map may remain unchanged from frame to frame while the light map may be updated by a dynamic light and the fog map may be changed by the moving camera
    - The texture coordinate and vertex correspondence can be different for each texture map

- Reference
  - See Real Time Rendering, 2nd ed., p146
  - See JL Mitchell’s page at http://www.pixelmaven.com/jason/
Multitexturing

- Multitexturing can
  - Save rendering passes
  - Allows more complex shading models than does the application of a single texture per pass
    - Example: compute a lighting model with expression $AB + CD$, where each variable represents a different color texture’s value
      - Impossible to evaluate without multitexturing or using off-screen rendering (stores images that can be combined later)
      - A multipass rendering algorithm could combine $AB$ in two passes and add $C$ in the next, but it could not fold in $D$ because $C$ would already have been added to $AB$. There is no place to keep $C$ separate from $AB$, since only one color can be stored in the frame buffer.
• Combine the results of these texture accesses, a **texture blending cascade** is defined that is made up of a series of texture stages.

![Diagram of texture blending cascade](image)

**Figure 29. Multitexture Texture Environments.** Four texture units are shown; however, multitexturing may support a different number of units depending on the implementation. The input fragment color is successively combined with each texture according to the state of the corresponding texture environment, and the resulting fragment color passed as input to the next texture unit in the pipeline.
Multitexturing

Two maps: a color map and a light map

Texture map

Light map
3D Texture Mapping

- Texture is determined by the intersection of the surface with the predefined 3D texture field.
- Texture field is obtained by procedure generation. Vertex coordinates are used to index a procedure (e.g. 3D noise function) that defines the 3D texture field for that point.

Advantages

- is successful at simulating turbulence (e.g., marble objects).
- eliminates mapping problems.
- objects of arbitrary complexity can receive a texture in a coherent fashion.
3D texture mapping in object space
3D Texture Mapping

3D mapping and 3D noise
Surfaces modulated by three-dimensional texture fields.

(left) Cube of cubes: the centre point of each cube is used to obtain a colour from the RGB cube. (below) Wood grain: harmonic functions (Fourier synthesis). (bottom left) Marble texture: equivalent to harmonic function plus noise. (bottom right) Using a three-dimensional noise function.